Introduction

MATLAB is an advanced interactive software package specially designed for scientific and engineering numerical computation. It is marketed by the Math Works, Inc. and is available in both professional and student versions. The MATLAB environment integrates graphics illustrations with precise numerical calculations, and is powerful, easy-to-use, and comprehensive tool for performing all kinds of computations and scientific data visualization.

MATLAB has proven to be a very flexible and usable tool for solving problems in applied mathematics, physics, chemistry, engineering, medicine, finance, and other application areas which deal with complicated numerical calculations. Along with the core program, there are a number of toolboxes that provide applications to specific disciplines, including control, telecommunication, signal processing and system analysis. It is one of the most popular mathematical software programs available, and is widely used in industry and education.

The fundamental features of MATLAB

MATLAB is controlled by commands, and it is programmable. There are hundreds of predefined commands and functions which can be further enlarged by user-defined functions. It is best to illustrate the fundamental features of MATLAB with a few examples. To launch MATLAB, just double click the icon and a 'MATLAB command window' will prop up.

Example 1
Unlike other programming languages, like PASCAL and C, in which a variable must be declared before it can be referred to later in the program, one can simply assign a value or matrix to a variable without prior declaration with MATLAB.

Enter the following commands and notice what responses appear in the MATLAB command window.

```
1. x = 10
2. x
3. x * 2.7
4. x / 2.5;
5. ans
6. x = x *2.56;
7. x
```

What do you think the function of a semicolon at the end of a command line is?
A concise manual on all the MATLAB operators can be accessed through the help window under the directory matlab\ops.

Example 2

Apart from a single number, MATLAB can also handle a matrix as well. Try out the following commands.

```
1. m = [11 12 13 14;21 22 23 24;31 32 33 34;41 42 43 44]
2. v = [1;2;3;4]
3. V = [1 2 3 4]
4. V*v
5. v*V
```

Remember that MATLAB is case sensitive, i.e. `v` is different from `V`. It should be noticed that the elements of each row are entered in sequence, with a space between the elements. (A comma may also be used to separate elements in a given row.) Each time a new row is desired, a semicolon is placed on the line. The matrix begins with a left-hand square bracket and ends with a right-hand square bracket. Thus, for the above examples, m is a 4-by-4 matrix, `v` is a column vector and `V` is a row vector.

In fact, MATLAB is built upon a foundation of sophisticated matrix software, in which the basic data element is a matrix that does not require pre-dimensioning. Therefore, MATLAB treats every variable as a matrix and any manipulations on the variables are matrix-based. The name MATLAB is derived from MATrix LABoratory.

Matrix Manipulations

MATLAB provides a number of matrix functions, among the most common and useful are inverse, determinant and eigenvalues/eigenvectors which can be called through the commands `inv`, `det` and `eig` respectively.

Example 3a

For a 4 x 2 matrix:

```
aMatrix = [5 10; 10 50; 30 10; 2 3]
```

The ‘colon notation’ can be used to select a column or row of a matrix.

To select the first column of `aMartix`, use

```
>> aMatrix (:, 1)
```

and the result is:

```
5
10
30
2
```

To select the third row of `aMartix`, use

```
>> aMatrix (3, :)
```

and the result is:

```
    30   10
```

For example, the inverse of a square matrix x is issued after entering the command `inv(x)`. All the available matrix functions in MATLAB are listed in the directory matlab\matfun which may be accessed through the help window. If you want a detailed description on how to use a particular command and any related commands, simply type `help command name`.

Example 3b

```
» help inv

 INV    Matrix inverse.
    INV(X) is the inverse of the square matrix X.
    A warning message is printed if X is badly scaled or
    nearly singular.

    See also SLASH, PINV, COND, CONDEST, NNLS, LSCOV.

 Overloaded methods
    help zpk/inv.m
    help tf/inv.m
    help ss/inv.m
```

Example 4

Two of the most widely employed methods for analyzing complex electric circuits are the mesh current method and the node voltage method. Both methods are rather general in that, in theory, most common circuits can be analyzed by either method. After the applications of KVL (or KCL) to the meshes (or nodes), a set of simultaneous equations is required to be solved. Suppose an electric circuit is analyzed with mesh current method and the following three mesh current equations are formulated.

$$11I_1 - 5I_2 = 45 \qquad (1)$$
$$-5I_1 + 10I_2 - 2I_3 = 1 \qquad (2)$$
$$-2I_2 + 13I_3 = -43 \qquad (3)$$

To solve this problem with MATLAB, these simultaneous equations are represented in matrix formulation as follow:

$$\boldsymbol{RI} = \boldsymbol{V} \qquad (4)$$

where $\boldsymbol{R} = \begin{bmatrix} 11 & -5 & 0 \\ -5 & 10 & -2 \\ 0 & -2 & 13 \end{bmatrix}$ $\boldsymbol{I} = \begin{bmatrix} I_1 \\ I_2 \\ I_3 \end{bmatrix}$ $\boldsymbol{V} = \begin{bmatrix} 45 \\ 1 \\ -43 \end{bmatrix}$

The mesh current ***I*** is then computed with the command `I = R\V` after defining the values of ***R*** and ***V***. The result from MATLAB should read like this:

```
I =

    5.0000
    2.0000
   -3.0000
```

Students are advised to try the command `I = V/R` instead and answer the following questions.

Q: Could you get the answer with the command `I = V/R` ? If no, why?
(Hint: Look up the difference between the operators ' \ ' and ' / ' from the matlab\ops directory. Then consider the matrix dimensions.)

Ans: With the reverse slash \ , the denominator appears on the left and the numerator appears on the right and vice versa with forward slash /. (Although in matrix manipulation, the actual process taking place is matrix inversion.)

Q: Could you use `inv` instead of the division operators to calculate the answer?

Ans: `I = inv(R)*V`

Graphical Output in MATLAB

Another powerful feature of MATLAB is data visualization. It has a wide range of 2- and 3-dimensional graphics commands which are flexible and easy to use. You can easily plot a set of ordered pairs, and one way to do it is to use the plot command. The following examples show you how convenient it is to generate a graph with MATLAB.

Example 5
To plot a graph with MATLAB, you must first enter the raw data in the form of either matrix or vector. The input data could be your experimental result, MATLAB generated simulation result or even a mathematical equation.

Let us make a plot of the following experimental data:

X	0.6	0.62	0.64	0.66	0.68	0.70	0.72	0.74	0.76
Y	0.0011	0.0023	0.0049	0.0106	0.0228	0.0493	0.1063	0.2295	0.4952

After entering the data, the command `plot(x,y)` generates figure 1.

In fact, figure 1 is a typical $I_c - V_{be}$ characteristics for a BJT transistor which is governed by equation 5:

$$-4 - I_c = I_s \exp(V_{be} / V_T) \qquad (5)$$

where I_s is the *saturation current* and V_T is the *thermal voltage*.

Figure 1 can also be plotted easily in MATLAB:

```
» vbe = 0.6:0.2:0.76;
» ic = 1e-13*exp(vbe/0.026);
» plot(vbe, ic, 'o');
```

The first command defines a variable vbe as a 1-by-9 row vector containing the data X. The second command works out the values of collector current according to equation 5. Finally, the plot command generates figure 2.

Figure 1 The vector Y plotted versus the vector X

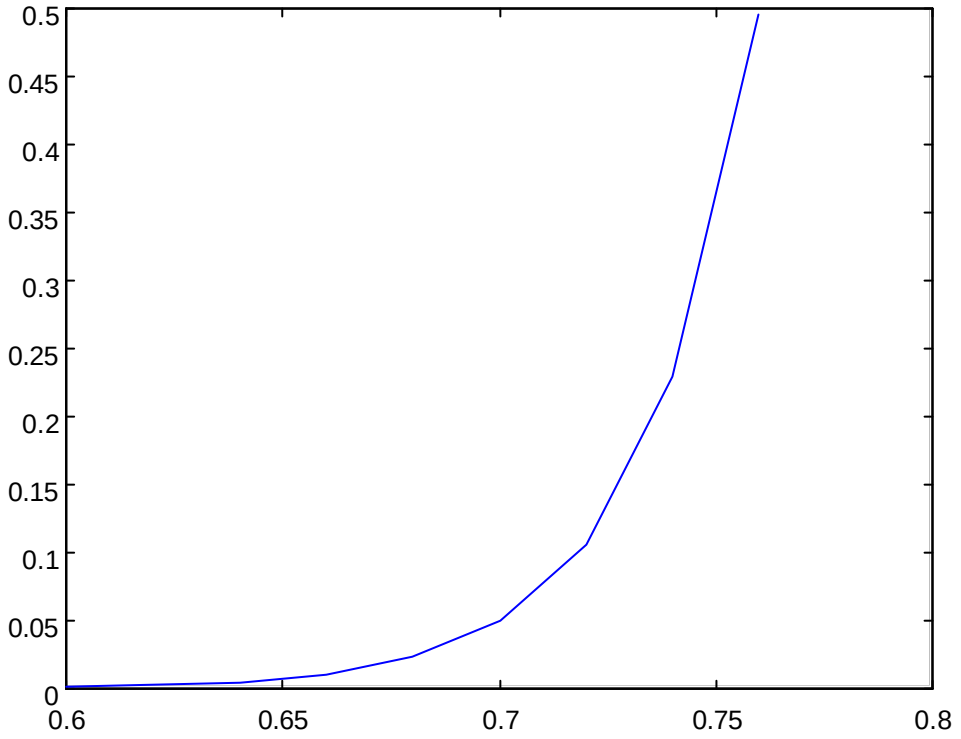
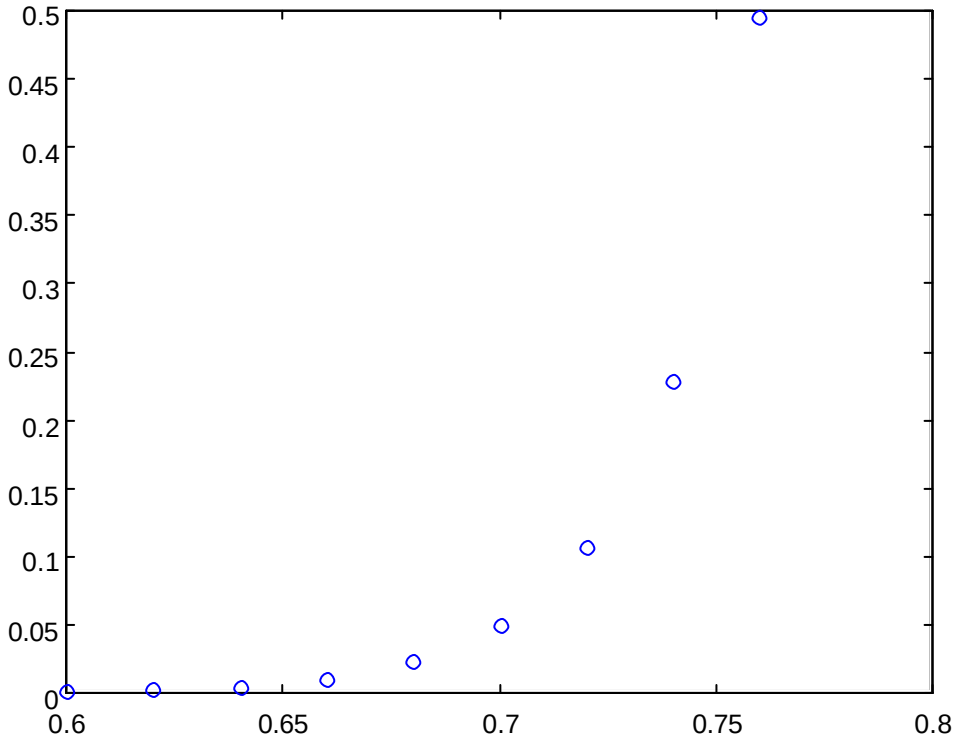


Figure 2 Ic - Vbe characteristics of BJT



There are many related commands on graph manipulation. The most frequently used ones include: `xlabel, ylabel, title, hold, subplot, zoom` and `grid`. You may use help to access detailed descriptions on these commands.

Students are advised to try the followings:

1. label the x- and y- axes and give a title to the graph
2. plot the graph with different range and/or resolution of V_{be}
3. plot, on the same graph, the I_c – V_{be} characteristics with different values of thermal voltage
4. use the subplot command to combine a few graphs in one figure

Exercise
Solve, graphically, the simultaneous equations y = sin(pi*x) and y = 0.5+0.5x.

Ans:

```
» x = 0:0.1:2;
» plot(x, [sin(pi*x); 0.5+0.5*x]);
```

A list of elementary mathematical functions can be found in the matlab\elfun directory through help window.

At the end of the session, you may want to save the variables that you have already defined. This could be easily done with the command `save`. The data can be retrieved with the command `load`. Try to use help to find out more about these commands.

Structure

The STRUCT command creates or converts a set of variables to structure array. It is represents as below:

```
aStruct = STRUCT('field1', VALUES1, 'field2', VALUES2, 'field3',
VALUES3, ...)
```

aStruct creates a structure array with the specified fields and values. The value arrays VALUES1, VALUES2, VALUES3 etc. must be cell arrays of the same size. Corresponding elements of the value arrays are placed into corresponding structure array elements. The size of the resulting structure is the same size as the value cell arrays or 1-by-1 if none of the values is a cell.

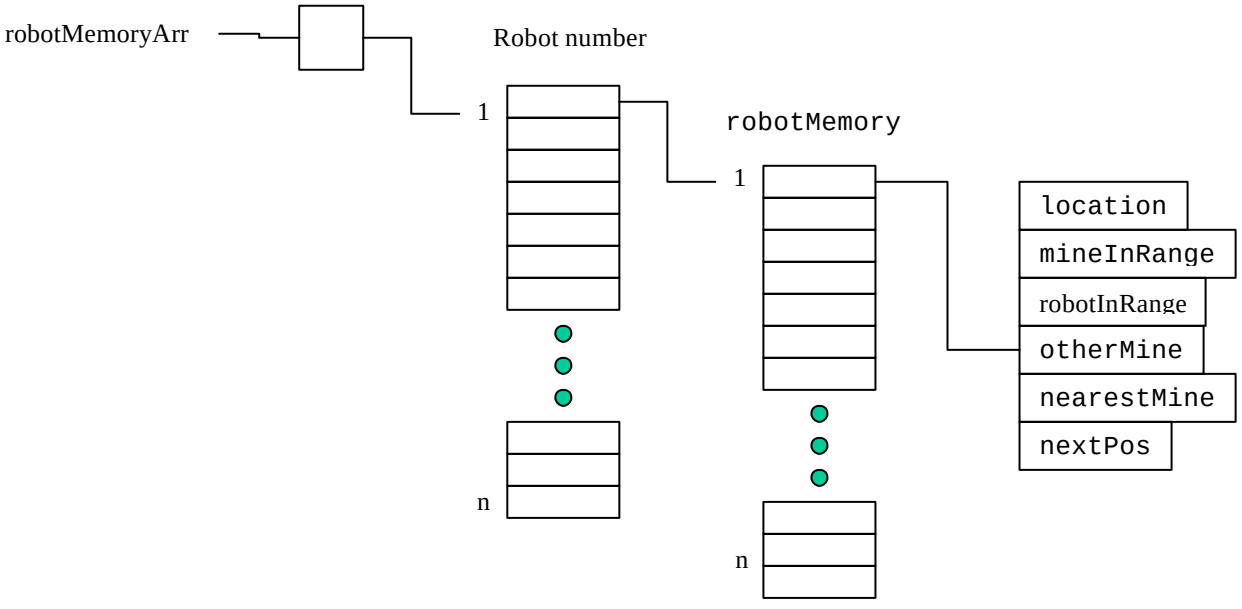
For example:

```
robotMemory = struct('location',[], 'mineInRange',[],
'robotInRange',[],'otherMine',[], 'nearestMine',[], 'nextPos',[]);
```

A structure called robotMemory is created with six fields which are

robotMemory
location[ ]
mineInRange[ ]
robotInRange[ ]
otherMine[ ]
nearestMine[ ]
nextPos[ ]

It represents the memory of a robot and describes all information obtained by a robot. A structure is assigned to each robot upon creation as, for MineDiffuse.m, robotMemoryArr is created to store every robot generated. Here, is the data structure:



Examples of assign value or make reference to `robotMemoryArr`:

- `robotMemoryArr(1).location = [10 5];`
  This is to assign a position of (10, 5) to the first robot

- `robotMemoryArr(j).nextPos = diffLocation(2,:);`
  This is to reference the next position of the j^{th} robot to be the value in the second row of the diffLocation matrix.

For Statement

- Create a loop from 1 to n:

```
for j=1:x
   ……..
end
```

- If we do not have a infinite number for how big the loop should be, we can use `size(variableA)` to tell the program when the loop should end. In this case `variableA` should be in the form of matrix.

```
for j=1:size(eightPos,1)
   ……..
end
```

  In this case, the range of the `for` loop is equal to the size of `eightPos` first column. For example, `eightPos` is a 8 x 3 matrix. The size of the `for` loop is equal to eight and the loop would execute from 1 until j reaches the value 8.

While statement

The general form of a `while` loop is:

```
while relation
     statements
end
```

If statement

The general form of a simple `if` statement is:

```
if relation
     statements
end
```

The general form of a simple `if-else` statement is:

```
if relation
     statements
elseif relation
     statements
else
     statements
end
```

Steady state response

In the context of electronics, a signal refers to the variation of voltage or current with time. Very often we may need to manipulate a signal in order to facilitate the subsequent operations on the signal. For example, in a conventional telephone system, a voice signal has to be amplified, filtered and modulated before it is transmitted along the telephone line. Among all these processes, amplification and filtering are the most common and fundamental in signal processing. In this session, you will learn how to use MATLAB to design a simple analogue filter and visualize its output characteristics.

Passive Filter

Resistors, capacitors and inductors are the most common passive circuit elements. Virtually any kind of analogue filters can be constructed with these three elements only. In sinusoidal steady-state, we may apply Ohm's law to all these three passive components, such that

$$V = ZI \qquad (6)$$

where Z represents the impedance of the circuit element.

Circuit element	*Impedance (Z)*
Resistor	R
Capacitor	$1/j\omega C$
Inductor	$j\omega L$

Table 1

In table 1 ω is the angular frequency equivalent to $2\pi f$ where f is the frequency in Hz. Armed with this generalization of Ohm's law, we may now treat capacitors and inductors as if they were resistors and apply all the conventional circuit theories to solve a circuit with capacitors and inductors. Say, for example, a capacitor C in series with a resistor R has a total impedance of $R+1/j\omega C$.

RC Low Pass Filter (LPF)

Try to analyze the circuit below and verify that the transfer characteristics, i.e. V_{out} / V_{in}, is given by equation 7.

$$V_{out} / V_{in} = (1/j\omega C) / [R + 1/j\omega C]$$
$$= 1/(1 + j\omega CR) \qquad (7)$$

Given that C = 1μF and R = 1kΩ, plot the *frequency response* of this low pass filter on a log-log scale.

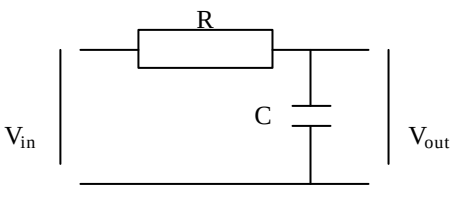


Figure 3 Simple low pass filter

(HINT: The frequency response of a filter is normally composed of two graphs. The first one is the magnitude variation with frequency and the other one is phase shift versus frequency. Students are required to plot at least the magnitude frequency response. Since the transfer characteristic is a complex function, you may want to use the `abs` command. In addition, i and j are automatically defined as $\sqrt{-1}$ in MATLAB. A log-log graph can be plotted with the `loglog` command. Remember to fully label the graph.)

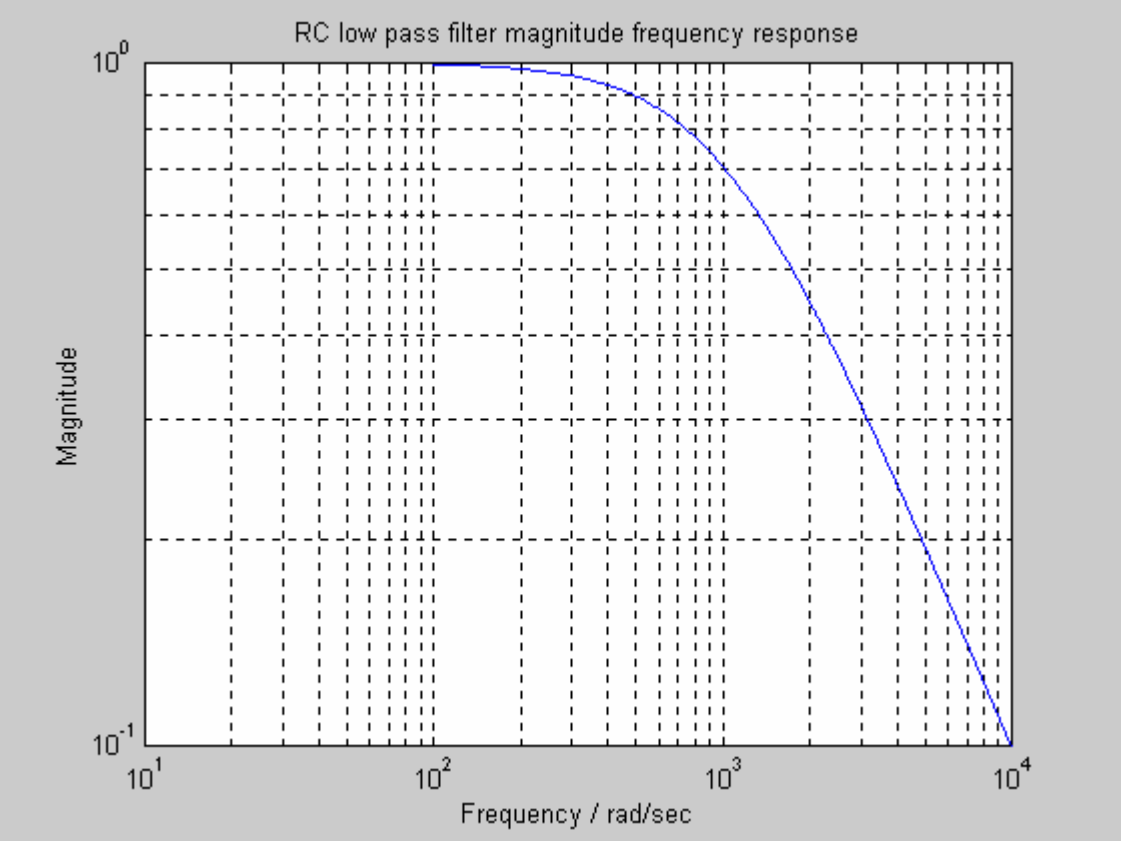


Figure 4 Frequency response of RC LPF

The cut-off frequency is defined as the frequency for which the transfer function magnitude is decreased by the factor $1/\sqrt{2}$ from its maximum value. What is the cut-off frequency of the filter? Could you identify from the transfer function what determines this cut-off frequency?

RL High Pass Filter (HPF)

Similar to the RC LPF, a high pass filter can be constructed from an inductor and resistor as shown in figure 5. Try to work out its transfer function.

$V_{out} / V_{in} = j\omega / [R/L + j\omega]$ (8)

Given that R = 1kΩ and L = 0.5H, plot the frequency response and estimate the cut-off

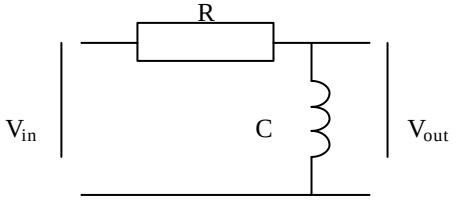


Figure 5 RL high pass filter

frequency. What determines the cut-off frequency?

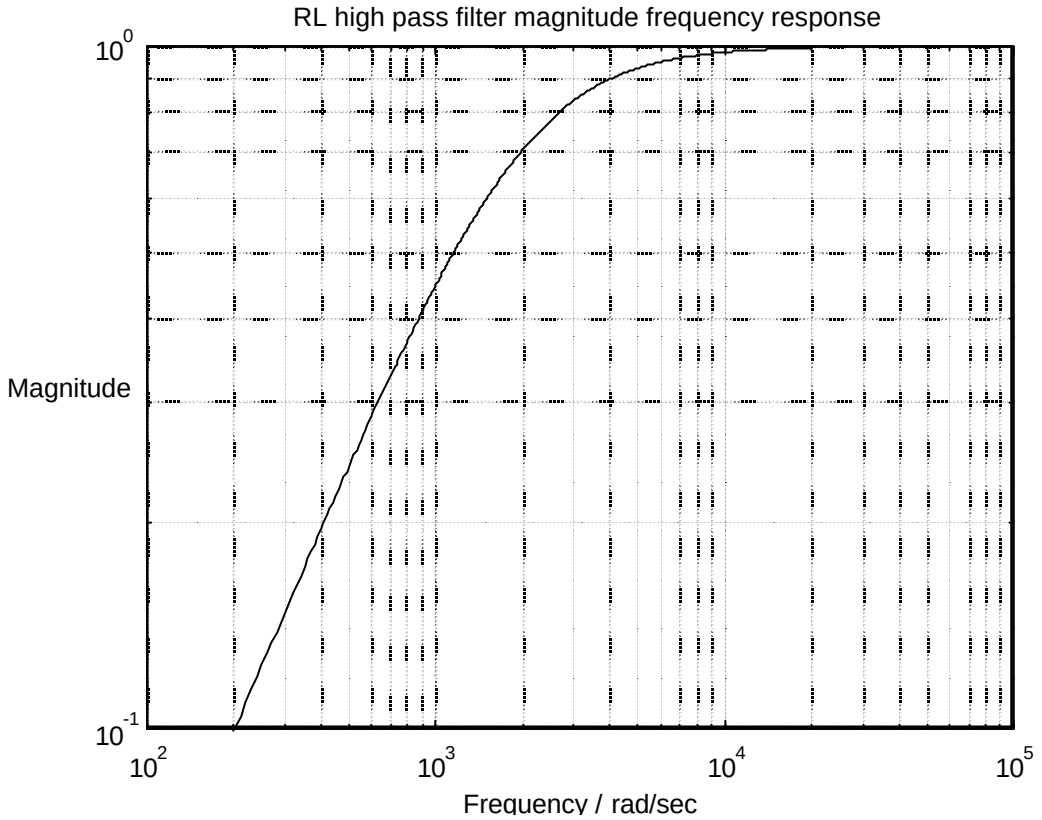


Figure 6 RL high pass response

RCL Band Pass Filter (BPF)

A band pass filter is formed from a high pass filter followed by a low pass filter, provided that the cut-off frequency of the HPF is lower than that of the LPF. What if the reverse is true? Figure 7 is an example of a RCL BPF, or a resonant circuit.

$$V_{out} / V_{in} = j\omega(R/L) / [(1/LC) + j\omega(R/L) + (j\omega)^2] \qquad (9)$$

The center frequency, f_c, of this BPF is given by $\sqrt{(1/LC)}$ and the bandwidth is equal to R/L. By selecting the component values appropriately, try to formulate a BPF with center frequency of 10kHz and a bandwidth of 1kHz. Plot the frequency response to verify your selections. Now,

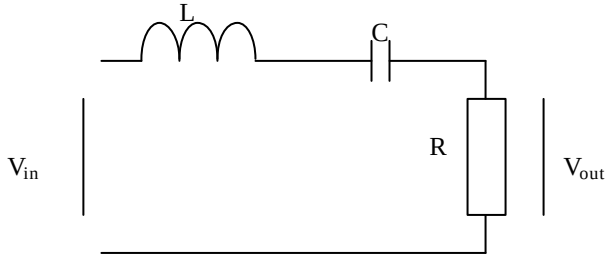


Figure 7 RCL series circuit

try to alter the resistance value and see what happens.
In fact there are enormous ways to design a filter with passive components only. One could make a high pass filter out of resistors and capacitors only, equivalently, a resistor and an inductor are enough to make a low pass filter. Different requirements and practical considerations make a specific configuration more suitable and desirable to a specific application. The examples described in this lab sheet are just a few of those. Could you design a high pass filter with a resistor and a capacitor? (It is really simple!)

s-domain analysis

Any signals could be represented by a combination of sinusoids with different amplitudes, frequencies and phase shifts. This is the fundamental of Fourier Transform. Fourier Transform is a mathematical tool to work out the frequency spectrum of a signal. Laplace transform is another mathematical tool to convert a time-domain signal or system to s-domain. To make life easier for you, we would just state that $s = j\omega$. Straight away, we could represent all the transfer functions derived previously in s-domain. For the low pass filter, we have $1/(1+sCR)$. With the RL high pass filter, the transfer function is $s/(s+R/L)$.

Pole and Zero

In general, any continuous systems, including filters, can be represented as a ratio of polynomials in s-domain, as in equation (10).

$$H(s) = K(s - z_1)(s-z_2)\ldots\ldots\ldots(s-z_n) / (s-p_1)(s-p_2) \ldots\ldots\ldots(s-p_m) \qquad (10)$$

Where K is the d.c. gain, z_1, z_2, ….. z_n are called the zeros and p_1, p_2, …..p_m are the poles. Clearly, a zero is the value which would make the transfer function equals zero and a pole is the value that would make H(s) goes to infinity. Take, for example, the RC low pass filter, its transfer function has a pole at $-1/CR$ whereas the RL high pass filter has a pole at $-R/L$ and a zero at $s=0$. Could you identify the significance of a pole to the frequency response of a filter?

To fully understand the significance of poles and zeros to the frequency response of a filter, you must try to plot the following expressions. (Hint: Convert back to $j\omega$ notation.)

1. $5 / (s + 1000)$
2. s
3. $s – 1000$

Discussion
When expressed as a ratio of two polynomials in s, the transfer function can be easily transformed into frequency domain and hence, the frequency response. A pole, p_1, in the transfer function causes the magnitude frequency response to roll off to 0.707 of its maximum value (or d.c. gain) at a frequency equal to p and continue to do so at a rate of 3dB/decade. A zero is the exact opposite of a pole. For multiple pole/zero

transfer function, the overall frequency response is a direct superposition of all the poles and zeros appearing in the transfer function expression. The following example will help you understand this idea.

Example 6
A second order band pass filter may be represented by:

$$H(s) = 10(s-100) / (s+1000)(s+10000) \qquad (11)$$

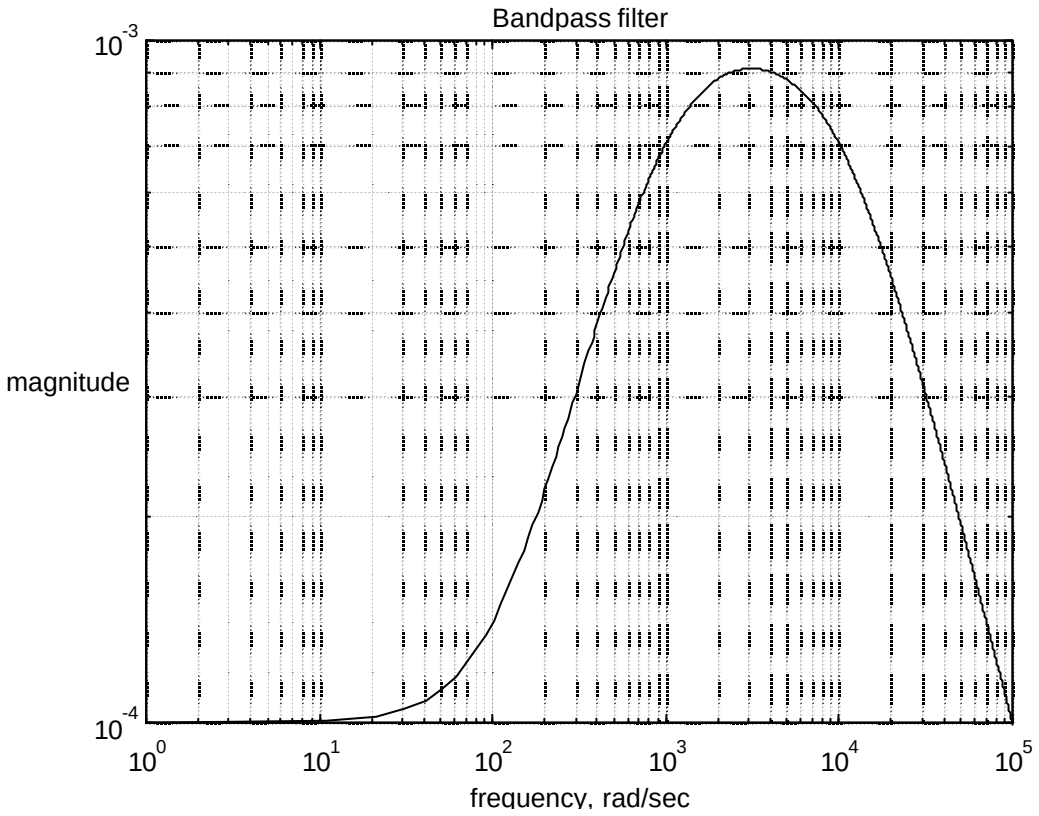


Figure 8 Band pass filter response

The zero at 100rad/sec causes the magnitude curve to rises until 1000rad/sec where the first pole cancels out the effect of the zero. At around 10,000rad/sec, it starts rolling off due to the second pole at 10000.

Filter design problem: Design a band stop filter with the stop band frequency in the range between 1kHz and 5kHz.

Discrete time signals

The counterpart of the Laplace transform for discrete time systems is the z-transform. The z-transform method of analyzing of discrete time systems is parallel to that of the Laplace transform method of analysis of continuous time systems, with some minor differences.

The response of a discrete time system, Y(z), to an input signal, X(z), is simply the product of the input signal and the system transfer function, H(z), i.e. $Y(z) = H(z)X(z)$. To visualize the frequency response, we can simply replace z with $e^{j\Omega}$ and plot the transfer function, similar to the continuous time systems.

IMPORTANT: $\Omega = 2\pi fT$ where T is the sampling period of the signal and $1/T = f_s$ is called the *sampling frequency*. Any discrete time signals must conform to the Nyquist sampling criterion which states that the sampling frequency must be at least twice the highest frequency component of the analogue signal to be sampled. Failure to follow the criterion causes aliasing, i.e. corrupting the original signal.

It follows that f/f_s must be less than or equal to ½. Since $e^{j\Omega}$ is periodic with a complete cycle of 2π, the magnitude response of a discrete time system must also be periodic and symmetrical about π. To verify this inference, the frequency response of a digital low pass filter is plotted.

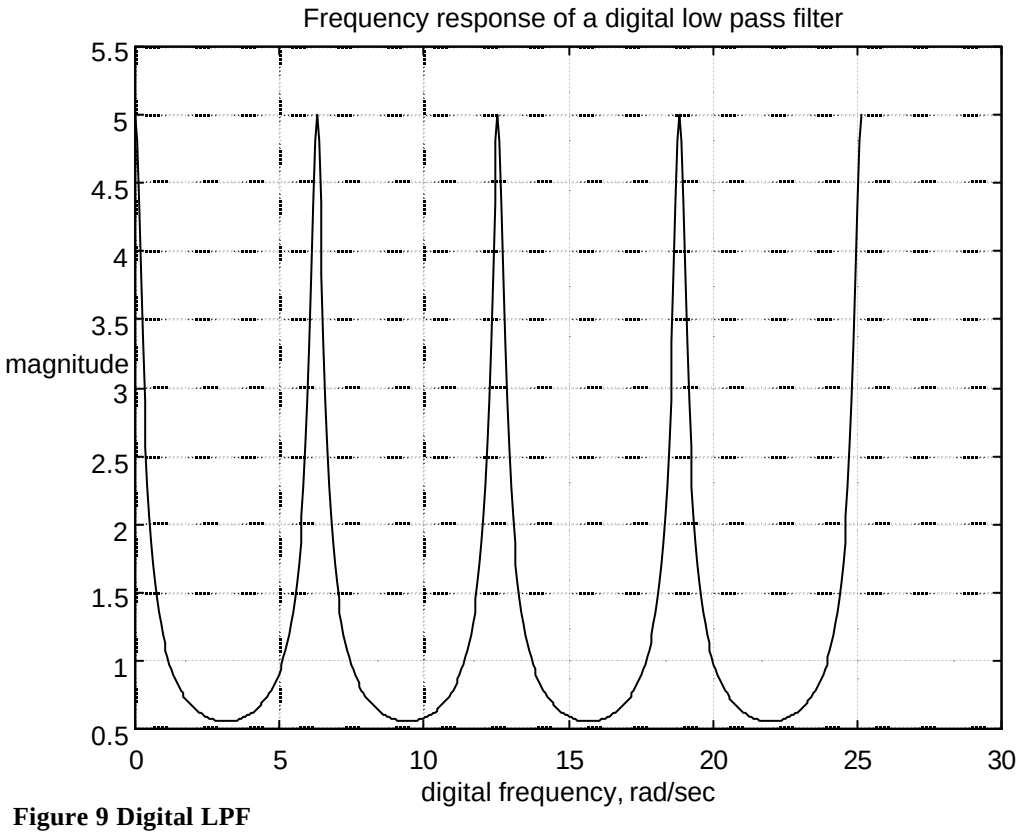


Figure 9 Digital LPF

The transfer function of this LPF is given by $H(z) = 1 / (1 - 0.8z^{-1})$. You must be familiar with this kind of expression, since it implies a pole at z = 0.8.

Design tips with poles and zeros placement

Given the specification of a digital filter, one can easily determines the transfer function of the required filter with pole-zero placement method. Starting with a unit circle in the z-plane (or complex plane), all the poles must lie within the unit circle for stability reason. The frequency response can be inferred from the position of poles and zeros within the unit circle. The following are a few key points:

1. To enhance the amplitude response at a frequency ω, we should place a pole as close as possible to the point $e^{j\omega T}$ (on the unit circle) representing that frequency ω.
2. Similarly, to suppress the amplitude response at a frequency ω, we should place a zero as close as possible to the point $e^{j\omega T}$ (on the unit circle) representing that frequency ω.
3. Placing repeated poles and zeros will further enhance their influence.
4. Total suppression of signal transmission at any frequency can be achieved by placing a zero on the unit circle corresponding to that frequency. This is the principle of the notch (bandstop) filter.

Design example

To design a high pass filter, we need to suppress the amplitude response at low frequency while enhancing it at high frequency. Figure 10 shows the desired position of poles and zeros in the z-plane. It is a third order high pass filter. Placing 3 zeros at z=1 largely suppress the gain at low frequency, while the poles at high frequency end enhance the amplitude there. It is worth noting that complex poles come in conjugate pairs.

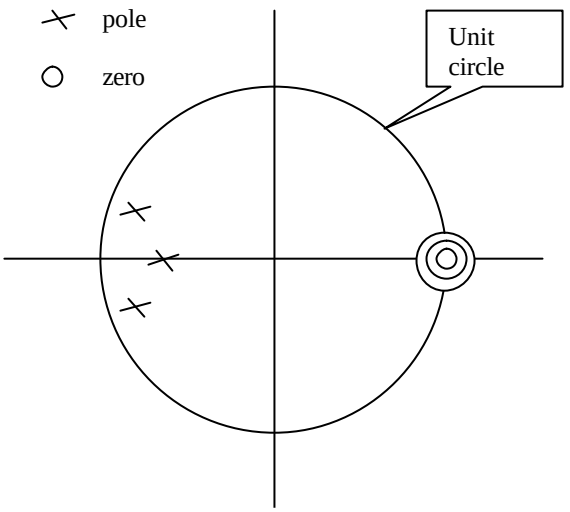


Figure 10 Pole-zero configuration of high pass filter in z-plane

One can easily plot the zeros and poles in z-plane with the MATLAB command **`zplane`**. To work out the associated frequency response, one may use the command `freqz`.

Try to plot the frequency response of this third order high pas filter. You may decide the exact locations of the poles and zeros. You may also find the commands `poly` and `roots` useful.

Implementation of digital filter

It has been stated that the output of a digital filter is simply the product of input and transfer function of the filter all represented in z-domain. To find the output response to an input signal x(n) in time domain of a particular filter, we have to convert x(n) into X(z) through z-transform and then multiply by the transfer function. The product Y(z) is taken back to time series by inverse z-transform.

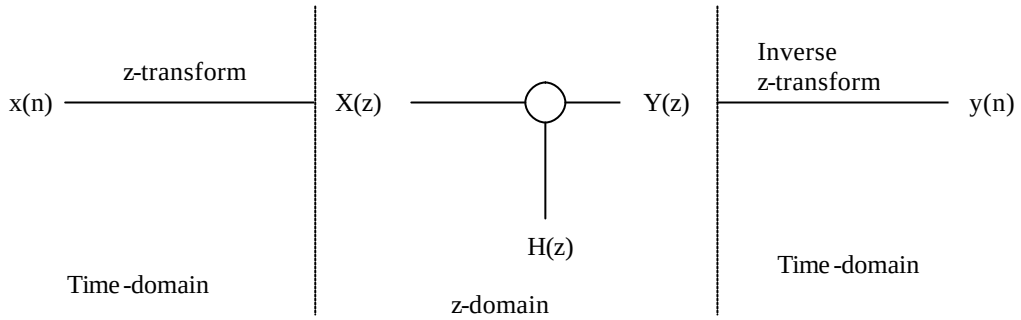


Figure 11 Flow diagram of signals

Figure 11 shows how the output time response y(n) is computed via the z-domain, given the system input signal.

Difference equation and transfer function

Difference equation is a general way to describe the present output of a system in terms of the present and/or past inputs and/or outputs. It is written in the form:

$$y(n) = \sum_{m=0}^{M} b_m x(n-m) - \sum_{k=1}^{N} a_k y(n-k) \qquad (11)$$

which shows that the present output value y(n) can be computed from the present and M past input values and N past output values. The coefficients b_m and a_k represent the corresponding weightings of the inputs and outputs respectively. In fact, a filter is defined by these coefficients. Clearly, one may readily implement a filter with any programming languages once given the values of the coefficients. However, the difference equation by its own does not, in general, provide any insight of the system concerned, like stability, frequency response and system function. Therefore, we need to convert it into the transfer function H(z) through z-transform, such that:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{m=0}^{M} b_m z^{-m}}{\sum_{k=0}^{N} a_k z^{-k}} \qquad (12)$$

You should be familiar with this equation, since it specifies the locations of poles and zeros of a filter. Conversely, given the transfer function of a filter, one may readily implement the filter described by the difference equation which is, of course, the inverse z-transform of the transfer function.

Reference

1. Digital Filters and Signal Processing (3rd), with MATLAB Exercises by *Leland B. Jackson*
2. Linear Systems and Signals by *B. P. Lathi*
3. Electric Circuits (5th), by *James W. Nilsson & Susan A. Riedel*